Hybrid Specialization: A Trade-off Between Static and Dynamic Specialization

Minhaj Ahmad Khan Henri-Pierre Charles
Denis Barthou
University of Versailles-Saint-Quentin-en-Yvelines, France.

Code specialization is a well-known technique used to produce more efficient code from a generic one. This technique is widely used, from high level languages to optimized libraries like FFTW. It works by substituting a formal input value by an effective value, and can be done either statically or dynamically. Static specialization makes use of data that is expected to be frequently used, whereas, dynamic specialization uses the actual values at run-time.

We propose a novel hybrid method combining both the static and the dynamic specialization.

With the code specialized for different values, the compiler is able to generate highly optimized code. This lets the compiler fully exploit the ILP provided by modern processors. Code specialization is most effectively performed at runtime due to the unavailability of input values. This information may be used for a wide range of optimizations and analyses; dependences may be simplified due to information related to array indices or other optimizations such as constant propagation, dead code elimination, loop unrolling and software pipelining can be performed. The specialization approach that we suggest proceeds in two steps performed at both static and dynamic compile times. In the first phase, the code is specialized for profiled values, and a template is selected through compile-time analysis. Consequently, the specializers for the templates are also generated after the analysis. The generation of specializers is followed by generation of code to perform activities required for cache coherence (in case of IA-64) for new specialized code. In the second phase, the template is transformed at runtime into the appropriate specialized version. A specializer (corresponding to a template), modifies a few instructions in the template to generate the correct code for runtime input data.

A wrapper code is also generated which redirects control to corresponding specialized or unspecialized version as shown in Figure 1.

The runtime specialization of values is also achieved by dynamic compilation systems like Tick C and off-line partial evaluators like Tempo[1] but they require the activities including code generation with memory allocation and yet these systems would require to keep different versions with

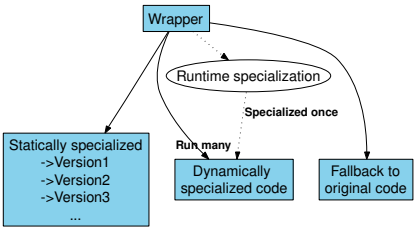

Figure 1. Runtime view of Wrapper

different optimizations. Moreover, the dynamic compilation systems require many calls to amortize the overhead. For the hybrid specialization approach, we avoid such time-consuming activities. The specialization is performed for a limited number of instructions in a generic binary template. To generate a single instruction, it incurs overhead of 9 cycles and 2 cycles over IA-64 and Pentium-IV processors respectively. This cost is very less as compared to dynamic compilation systems such as Tempo or Tick C which require 100 to 800 cycles to generate a single instruction. This template is generated during static compilation and is highly optimized since we expose some of the unknown values in the source code to the compiler. The template is then adapted to new values during execution thereby avoiding code explosion as in other existing specializers. We have implemented hybrid specialization in HySpec [2] framework obtaining performance improvement up to 50% for different DFT sizes of `FFTW` and up to 27% for some `SPEC` benchmarks.

References

[1] C. Consel, L. Hornof, R. Marlet, G. Muller, S. Thibault, and E.-N. Volanschi. Tempo: Specializing Systems Applications and Beyond. *ACM Computing Surveys*, 30(3es), 1998.

[2] M. A. Khan, H.-P. Charles, and D. Barthou. Reducing code size explosion through low-overhead specialization. In *Proceeding of the 11th Annual Workshop on the Interaction between Compilers and Computer Architecture, Phoenix*, February 2007.

16th International Conference on
Parallel Architecture and Compilation Techniques (PACT 2007)
0-7695-2944-5/07 $25.00 © 2007 IEEE